Quantum Search with Wildcards

Alec Landow and Liam Adams

November 17, 2020

1 Motivation

Search with wildcards or finding patterns in a string is a very important problem with applications in DNA sequencing, data streaming, and search engines[1]. When searching for patterns one can either search for an exact match or the closest match. Finding an exact match of a pattern classically can be solved in $O(N+M)$ time where N is the length of the input string w and M is the length of the pattern p to be searched for. Finding the closest match is actually more difficult, with a classical bound of $O(MN)$.

Finding the closest match is a more practical problem for domains in which the data can mutate over time, such as genetics[2]. DNA sequencing is a major application for quantum computing, and the algorithm we implemented for closest match, along with wildcard search, is applicable to that area of study.

Quantum computation provides the possibility to speed up the classical solution by using the properties of quantum mechanics so that we don't have to scan the full input string w to find the solution. Today's quantum computers are still noisy, but by representing the problem instance as quantum states we can find the solution with some probability. If we run the algorithm enough times we can be certain that the answer with the most occurrences is the correct answer.

The instances of the problem we considered are very small, such that it could be run on a simulator in a classical machine or a small quantum computer. However the algorithm can be run as it is on large problem instances to provide a real advantage over the classical algorithm once quantum computers with more qubits come online in the future.

2 Methodology

We based our implementation on that of Mateus and Omar[2]. This algorithm was designed for pattern matching in a string without wildcards. We made an adjustment in order to search with wildcards. This algorithm returns the position of the closest substring to a pattern p with non negligible probability in $O(\sqrt{N})$ queries, where N is the size of the input string w. Matching the closest substring rather than returning an exact match is a more difficult problem.

The algorithm is designed such that any number of patterns can be searched for on w with no additional overhead. The same circuit components can be reused as long as w and the alphabet w uses stays the same. The key to this is constructing oracles for every member of the alphabet during preprocessing, so when a pattern is identified to search the necessary oracles can be pulled in. This makes the circuit more modular which speeds up subsequent searches as it makes building the circuit for a particular pattern p much simpler.

We chose to work with an alphabet of 0 and 1 initially. However, we then extended the algorithm to larger alphabets, as we will explain below. The algorithm consists of three main components: the initial state, the oracles, and amplitude amplification (diffuser).

At a high level, the initial state is constructed and the necessary oracles are added to the circuit. The oracles invert the phase of our quantum states if a pattern character is present in w. Then the standard Grover diffusion is applied which amplifies the amplitudes of the states containing pattern characters.

The entire process is run $O(\sqrt{N})$ times. During each iteration, a random pattern character and its corresponding quantum register/state is chosen. The oracle for this character is then applied to that state and diffusion is then performed. This element of randomness allows us to find the closest match rather than the exact match. On average a starting position followed by M'< M matches will have the oracle applied $\frac{M'}{M}$ times, and a starting position with a complete match will have the operator applied M times. Therefore the amplitude for the state representing the start of the substring with the most characters in the pattern will be measured.

Finally, a measurement of the first register representing the first character in the pattern is taken, and the state with the highest number of occurrences should give the correct answer for the starting position of the pattern in the string.

2.1 Initial State

The algorithm outputs the starting position of the pattern in the string w, so we need our qubits to represent the indices of w. For a pattern of size M and string of size N, there are only $N - M$ positions for which the pattern can start. We need a quantum state for each character in the pattern to represent the characters' positions. We created a quantum register for each of the M characters in p, with each register containing $s = \lceil log_2(N - M + 1) \rceil$ qubits. Therefore each register can represent 2^s states or positions in the string w.

Now the registers representing the characters in the pattern need to be entangled so that the position of the first character is one before the second character and so on. Below is a picture of the initial state for our circuit with pattern size of 2 and input string 8.

The Hadamards followed by the CNOT gates from the first register to the second create a Bell state which is entangled. The CNOT with the empty circle on the control bit means flip if 0 rather than 1. As an example you can see if the first register is 000, then the bottom register will be 001. We are only

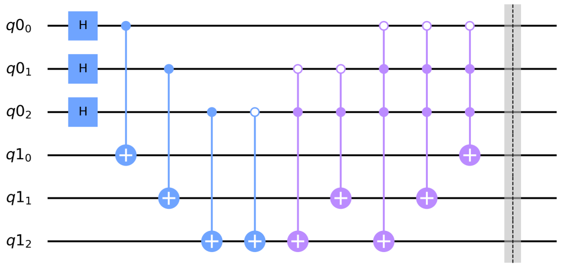

Figure 1: Pattern Size = 2, Input String Size = 8

interested in measuring the first register, so the Hadamards are only applied there to create a superposition of all states to measure.

2.2 Oracles

Now that we've created the initial state, we need to construct the oracles that can tell us whether a particular character in the string matches a pattern character. As we mentioned above, once the input string and alphabet is known, we can construct all the oracles and select the appropriate oracles for particular patterns. The oracles we implemented are very similar to the oracles in Grover's algorithm[3]. If a pattern character matches a position in the input string, we invert the phase of the state representing the pattern character.

We tried two strategies for the oracle which we called single oracle or many oracles. In single oracle we create a $2^s x 2^s$ identity matrix for each character in w, with a -1 in a diagonal entry if the character occurs in that position. So if a character occurs at position 3 and 5 in w, entries (3,3) and (5,5) in the matrix will be -1.

For the many oracles strategy, each matrix only has a single -1 in it. So in the previous example we would have 2 matrices rather than one, with -1 in the appropriate position. Once the matrices are constructed we convert them to unitary operators using the Qiskit API[4] so it can be used in the quantum circuit.

In both strategies we represented the wildcard character as a diagonal matrix with all -1 entries, so the state representing the wildcard character always has its phase changed. Also in both strategies, we only use the oracle matrices for the pattern we're interested in. Meaning for the randomly chosen pattern character as described in the Methodology section, we apply the respective oracle for that character to the register representing that character.

2.3 Diffuser

Once the oracle has been applied to the quantum register, we then apply Grover diffusion to amplify the amplitude of its state so that when we measure the registers the most likely measurement will come from a pattern character present in the string w. We found that it was better to only apply diffusion to the first register representing the index of the start of the pattern, since that is the register we're interested in measuring.

Diffusion works by applying a reflection on the state whose phase was just inverted by the oracle. The plane of the reflection is the 2D space spanning the orthogonal vectors w representing the state we're looking for and our current state s minus w, denoted s'. The matrix is constructed using the following equation

$$2\left|s\right\rangle\left\langle s\right| - I \tag{1}$$

which will flip our vector with negative phase about the vector $|s\rangle$ giving it a positive amplitude larger than it had before.

This matrix is again converted into a unitary operator for use in the circuit and applied to the first register. We also tried an alternative Diffuser implementation constructed using gates, derived from the qiskit textbook[3].

Once diffusion is performed we measure, and repeat $\sqrt{N}$ times. In the end we choose the state from the first register that was measured the highest number of times as our answer.

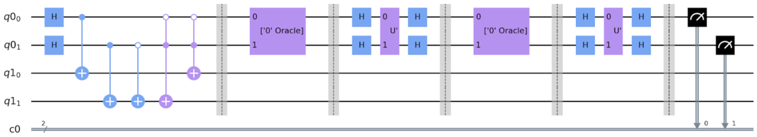

Figure 2: Input: 0101, Pattern: 01

3 Solved Issues

Below is a list of contributions for each member of the team.

Solved Issue	**Member Responsible**
Development of Skeleton Program	Both
Initial State	Alec Landow
Oracle	Alec Landow
Diffuser	Alec Landow
Alternative Implementation	Liam Adams
Final Report	Liam Adams

4 Results

Our test cases were drawn from an alphabet of {0,1} or [a-z]. The tests consisted of patterns with wildcards, single character patterns, two character patterns, and three character patterns. Our input strings w were either of length 4 or 8.

Our algorithm is not 100% correct but we do get some promising results. More often then not we are able to detect at least one occurrence of a pattern in a string. A fail means we did not detect any patterns and match means we were able to find at least one pattern.

Detecting patterns in the binary alphabet was difficult because there are so many overlapping patters, and the diffusor is applied so frequently that all the indices end up having their amplitude amplified. We acheived the best results for the Latin Alphabet Single Character because the alphabet is sparse enough that it is easy to determine a single pattern. Finding a single character actually is just the standard Grover's algorithm, and our oracles and diffuser are the same as those used in Grover's.

For the Latin Alphabet Multiple character we focused on exact matches within the string, in which the pattern only occurs once. Mateus and Omar[2] mention that the theoretical probability of finding an exact match in this scenario is at least $\frac{1}{4}$ without proof. Our results follow closely with this benchmark. Intuitively one would assume that a pattern that only occurs once in the string would be easier to detect, as there would be no amplification from other pattern occurrences. However our algorithm is randomized such that a position in the string that contains a character in the pattern, but is not part of the overall pattern will be amplified. This could contribute to noise in results.

Binary Alphabet All Patterns		
	Many Oracles	Single Oracles
Gate Diffuser	Fail: 44.65% Match: 55.35%	Fail: 49.74% Match: 50.26%
Matrix Diffuser	Fail: 44.13% Match: 55.87%	Fail: 47.94% Match: 52.06%

Latin Alphabet Single Character		
	Many Oracles	Single Oracles
Gate Diffuser	Fail: 23.33% Match: 76.67%	Fail: 26.67% Match: 73.33%
Matrix Diffuser	Fail: 26.67% Match: 73.33%	Fail: 15.83% Match: 74.17%

Latin Alphabet Multiple Character		
	Many Oracles	Single Oracles
Gate Diffuser	Fail: 65.45% Match: 34.55%	Fail: 71.36% Match: 28.64%
Matrix Diffuser	Fail: 70% Match: 30%	Fail: 69.55% Match: 30.45%

We achieved the best performance using the many oracles and matrix diffuser approach. The many oracles approach may be more in the spirit of the approach outlined in the paper. When a pattern character is randomly chosen, it applies the oracle to each occurrence of the character rather than to just one occurrence. This results in more of a phase change to that character and hence a greater amplitude amplification.

There wasn't much of a difference between the gate and matrix diffuser, as there are examples of a high match percentage for both. The gate based diffuser is more efficient however, as it doesn't rely on the qiskit compiler to generate gates from a matrix.

4.1 Optimizing for a Quantum Computer

The results above are all taken from the qiskit simulator. We did run our algorithm on a real quantum computer in IBMQ and achieved similar results.

For $N = 8$ and $M = 2$, each register has $s = \lceil log_2(N - M + 1) \rceil = 3$ qubits. Therefore we need $M * s = 6$ qubits. IBMQ has many options for machines of this size. For our algorithm in particular we need to entangle all of our registers which uses CNOT gates heavily, so it would be best to choose a machine with a low CNOT error rate.

From a topological perspective our diffuser is applied mainly to the first register, so our first register should be placed in a region with a high density of qubits. Also, the first register is entangled with the second, second to third and so on. We don't need to connect registers that aren't neighbors, so when assigning qubits to the registers we should place the qubits in neighboring registers close to each other.

5 Conclusion

In conclusion our implementation of search with wildcards showed some promising results but was not 100% correct. Possible future improvements include applying the diffuser to all registers, constructing the oracle from gates, extending the alphabet, and using ancilla qubits.

The biggest challenge in the algorithm was implementing the oracle. In the paper the authors describe the unitary operator for the oracle as a transposition operator using Gray codes, in which states can only be transformed to one state up (0 to 1, 1 to 2, etc). Additionally the authors referenced the usage of ancilla qubits in the oracle. Further research is needed to understand and implement these concepts.

References

[1] V. Menon and A. Chattopadhyay. Quantum pattern matching oracle construction, 2020.

[2] P. Mateus and Y. Omar. Quantum pattern matching, 2005.

[3] Qiskit tutorial on grover's algorithm. https://qiskit.org/textbook/ch-algorithms/grover.html.

[4] Qiskit 0.23.0 documentation. https://qiskit.org/documentation/index.html.